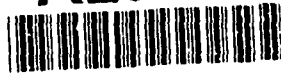


AD-A268 827

1.L. (2)



NASA
Technical Memorandum 106239

AVSCOM
Technical Report 92-C-020

Distributed Simulation Using a Real-Time Shared Memory Network

Donald L. Simon
Propulsion Directorate
U.S. Army Aviation Systems Command
Lewis Research Center
Cleveland, Ohio

Duane L. Mattern
Sverdrup Technology, Inc.
Lewis Research Center Group
Brook Park, Ohio

and

Edmond Wong and Jeffrey L. Musgrave
Lewis Research Center
Cleveland, Ohio



93-20453



2005

July 1993

NASA

APPROVED FOR STATE
Approved for public release
Distribution Unlimited



93-20453-8

Distributed Simulation Using a Real-Time Shared Memory Network

Donald L. Simon
Propulsion Directorate
U.S. Army Aviation Systems Command
Lewis Research Center
Cleveland, Ohio 44135

Duane L. Mattem
Sverdrup Technology, Inc.
Lewis Research Center Group
Brook Park, Ohio 44142

Edmond Wong and Jeffrey L. Musgrave
Lewis Research Center
Cleveland, Ohio 44135

ABSTRACT

The Advanced Control Technology Branch of the NASA Lewis Research Center performs research in the area of advanced digital controls for aeronautic and space propulsion systems. This work requires the real-time implementation of both control software and complex dynamical models of the propulsion systems. We are implementing these systems in a distributed, multi-vendor computer environment. Therefore, a need exists for real-time communication and synchronization between the distributed multi-vendor computers. A shared memory network is a potential solution which offers several advantages over other real-time communication approaches. In this effort, a candidate shared memory network was tested for basic performance. The shared memory network was then used to implement a distributed simulation of a ramjet engine. The accuracy and execution time of the distributed simulation was measured and compared to the performance of the non-partitioned simulation. The ease of partitioning the simulation, the minimal time required to develop software for communication between the processors and the resulting execution time all indicate that the shared memory network is a real-time communication technique worthy of serious consideration.

Accession For	
NTIS	CRA&I <input checked="checked" type="checkbox"/>
DTIC	TAB <input type="checkbox"/>
Unannounced <input type="checkbox"/>	
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

Nomenclature

ACTB	Advanced Control Technology Branch at NASA Lewis Research Center
CFD	Computational Fluid Dynamics
CIM	Control, Interface, and Monitoring computer
Frametime	Physical time that it takes to complete one simulation integration time step.
ICS	Intelligent Control System
IFPC	Integrated Flight and Propulsion Control
LeRC	Lewis Research Center
NASA	National Aeronautics and Space Administration
NASP	National AeroSpace Plane
Real-Time	Simulation time corresponds to actual physical time
Scalability	The ability to easily increase the number of nodes in a finite volume mesh of a CFD problem.
Scaled Time	Simulation time equals a linear function of physical time.
Speedup Ratio	Ratio of the amount of time that is required to run a simulation on multiple CPU's versus the time required to run the simulation on a single CPU.
SSME	Space Shuttle Main Engine
Time step	Simulation time per integration step

Introduction

As dynamic models for aeronautic and space propulsion systems become more accurate and more complex, alternative computer hardware configurations are being considered to address the increased computational burden. The consensus appears to be that multiprocessing will be required to provide the additional computational effort. Any multiprocessing scheme will require communication between the various processors. There are advantages and disadvantages to the various multiprocessor configurations because they solve different types of problems. For example, in "Grand Challenge" [1] CFD type problems, where the accuracy of the model is determined by the number of finite volumes that make up the simulation grid, scalability is an important factor and this is why massively parallel computers like the Intel HyperCube have been developed. In the Advanced Control Technology Branch (ACTB) at NASA Lewis Research Center (LeRC) our interest is in real-time simulations that involve computer hardware from a variety of vendors where the inter-machine communication is becoming a significant part of the overall simulation development effort. Any future simulations will likely require additional computing power but still must interface with the existing hardware environment. Thus, in the search for a solution to the problem of communication between computers from different vendors used in a real-time, distributed simulation, this paper

evaluates a shared memory, real-time fiber optic network. Although multi-vendor computer integration is required, the distributed simulation evaluation was performed on a test bed hardware consisting of computers from a single vendor.

The paper is organized as follows: 1) a description of the simulation environment in the ACTB at NASA Lewis Research Center and the types of propulsion systems currently being simulated are described; 2) various interprocessor communication techniques are listed and the shared memory network is described; 3) the hardware testbed setup to evaluate the shared memory network which is comprised of three Intel 386-20MHz machines is described and node to node shared memory communication times are summarized; 4) the software test application, the ramjet engine model, is described and the partitioning of this simulation is discussed; 5) the partitioned simulation results and subsequent speed up due to distributed processing are presented; 6) finally, the paper closes with a summary of the results of using a shared memory network for distributed real-time simulations.

Propulsion System Simulation Requirements

Currently, the ACTB has three main research programs: the preliminary control design and test for the government baseline engine for the National AeroSpace Plane (NASP) [2]; the development and testing of a control design method for Integrated Flight and Propulsion Control (IFPC) [3]; and the development and test of an Intelligent Control System (ICS) for the Space Shuttle Main Engine (SSME) [4]. These three programs use some of the same simulation computer equipment, yet none of the simulation hardware configurations are identical. Figure 1 shows the NASP hardware configuration consisting of one Applied Dynamics International (ADI) AD100 computer containing the engine simulation, an Intel 486-33MHz Control Interface and Monitoring (CIM) unit which implements the control laws, and an EAI analog computer simulating the sensor dynamics. The interfaces in Figure 1 are all analog. Figure 2 shows the IFPC hardware configuration, consisting of an AD100 computer which contains the airframe and engine simulation, an Intel 386-20MHz Control Interface and Monitoring (CIM) unit which implements the integrated control laws, a 486-20MHz development station, image generator, and displays for the flight simulator visual projection system, and a cockpit for pilot in the loop evaluation. The communication paths between the CIM unit and the AD100 are analog, while the communication between the AD100 and the 486 machine is via a parallel digital interface

(DR11). The simulation testbed developed at LeRC for proof-of-concept of an ICS is shown schematically in Figure 3. The 486-33MHz CIM unit implements the control laws and provides the necessary coordination between the diagnostic system and the rocket engine controller [3]. The AD100 provides real-time operation with multichannel analog I/O for a rocket engine simulation complete with failure mode models. The Vaxstation 3500 runs a real-time rule based expert system shell called G2TM developed by GENSYM. A special purpose LISP machine, the TI Explorer II-Lx, provides a flexible object oriented environment for the development and implementation of a user interface for the ICS. Finally, a personal computer with an ANZATM board developed by HNC, Inc. implements the neural network algorithms used in the ICS. The CIM unit, AD100, and PC are interconnected with analog links providing real-time data transfer. A DR11 interface provides communication between G2TM and the CIM unit and a local area network provides the necessary data for the user interface.

In a tightly coupled multiprocessing system, all processors reside in the same computer chassis and communicate over the computer bus. A loosely coupled multiprocessing system on the other hand consists of distributed computer systems interconnected in some manner to allow communication. The three configurations shown in figures 1-3 are all loosely coupled systems. The machines in these simulations must be connected and synchronized to properly represent the physics and also for data collection, since no one machine collects all the data. The point of Figures 1-3 is to show that a variety of configurations are used at any one time, that these configurations change over time as the projects in the ACTB change, and that each new project brings with it some new computer hardware that has to be integrated into the existing environment. In this environment, the machine to machine communication can become a significant part of the simulation development time.

Interprocessor Communication Methods

There are a variety of interprocessor communication techniques that have been used for real-time multiprocessing. In reference 5, a full-duplex, asynchronous, 20-Mb/s serial channel for linking T800 transputers is described. Reference 6 examines the use of the Ethernet LAN and the Fiber Distributed Data Interface (FDDI) protocol for the real-time simulation of multiple aircraft. In reference 7, a bus-based shared-memory multiprocessor architecture was used for the parallel implementation of a real-time control system for a turbojet engine. All local

memory was available to the other processors over a VME bus, which allowed all the processors to compute without contention as long as they did not require a shared resource. A speedup ratio of 3.38 with four processors was obtained. The ACTB's past experience with parallel processing is displayed in Figure 4 [8]. Figure 4 shows three CPU's which were all housed in one 18 slot Multibus chassis. The data was transferred between the CPU's through dual-ported shared memory and synchronization between the CPU's was achieved through interrupts.

None of these interprocessor communication techniques are satisfactory for the purposes of the ACTB. As described in the previous section, the ACTB requires real-time communication between multiple distributed computers from a variety of vendors. The distributed message passing techniques allow for the integration of different vendors' CPU hardware and permit the physical separation of CPU's but not necessarily the real-time response or synchronization required, usually because of the software protocol overhead. The shared memory techniques can minimize delay and allow synchronization, but usually don't support the physical distribution or multi-vendor architectures required, and the contention for access to a bus can be a bottleneck. Both methods can require a significant software effort to code the communication and synchronization required between processors. Because of these shortcomings, we decided to search for a solution better suited to the needs of the ACTB.

Replicated Shared Memory Network

References 9 and 10 compare shared-memory architectures to message-passing networks and describe a replicated shared-memory network as an alternative. In a replicated shared memory network each computer on the network has a network interface card and its own local copy of shared memory. The application program uses the shared memory in exactly the same fashion as any other area of memory. However, any data written to shared memory is automatically transferred around the network and duplicated in the same shared memory locations of all the nodes. The transmission of data is performed by the network interface card and occurs transparently to the application program. Typically, the only application software overhead will be in initializing the network interface card and defining the variables to be shared between the distributed applications which reside in shared memory.

The replicated shared memory network offers several advantages over traditional distributed processing architectures. In message passing local area networks much of the transmission delay is due to the lengthy software protocol overhead required to transmit data, receive data, and detect and recover from errors. The replicated shared

memory network on the other hand implements all of the above functions in hardware allowing them to occur transparently to the application program which saves valuable CPU time. The replicated shared memory network also allows interrupts to easily be transmitted and received over the network which makes synchronization straight forward. The replicated shared memory network offers advantages over shared memory techniques as well. It allows communication between CPU's from multiple vendors, it permits physical distribution of the CPU's, and allows communication between a large number of CPU's.

The real-time replicated shared memory network product used in the demonstration described in this paper is the SCRAMNetTM network from the Systran Corporation [9]. The SCRAMNet network can support up to 256 nodes and each node can be separated by a distance of up to 700 meters. A SCRAMNet network interface card is required at each node on the network. This card consists of the shared memory, data filter, network interface, host adapter, and network control logic. The individual nodes are connected over a dual channel fiber optic ring network with a bandwidth of 150 Mbits/sec. As mentioned previously, whenever a local host writes to shared memory, the written value is transmitted to the same shared memory location in all nodes on the network. The data filter is the portion of SCRAMNet logic which checks the value that the local CPU is writing to shared memory and compares it to the value already stored at that particular memory location. If the two values are not identical, then the new value is updated around the network. Otherwise, no network transmission occurs. The data filter helps to eliminate unnecessary traffic on the network. The programmer has the option of disabling the SCRAMNet data filter so that all writes result in a network transfer regardless of the previous value stored in the shared memory location. The network interface hardware implements the network protocol, reads data from the network, writes it in shared memory, and transmits writes by the local CPU to other nodes on the network. SCRAMNet uses a register insertion network protocol, which is specifically designed for real-time data transfer and real-time error recovery. All message packets transmitted over the network are 82 bits in length, of which 32 bits is data and the rest is control, status, and address information. This fixed packet length protocol allows the SCRAMNet network to have deterministic transmission delays as opposed to variable packet length message passing protocols. Nodes can transmit simultaneously on the network, and all nodes have equal priority for network bandwidth. If the transmitted packet does not propagate around the network and return to the transmitting node in a set amount of time or if upon returning to the transmitting node a parity error is found, the message is re-transmitted. The SCRAMNet

documentation states that the amount of time consumed by the error detection and re-transmission process obeys the following formula:

$$\text{Time} = [\text{Number-of-Nodes} \times (496.7 \pm 296.7) + (1173.4 \pm 503.4)] \text{ nanoseconds}$$

The maximum time is the upper bound for the worst case scenario and will rarely occur. Most errors will be detected and retransmitted at a time closer to the minimum time.

The host interface establishes communication between the local CPU and shared memory over the local computer bus. The host interface will also interrupt the local CPU upon network writes to an interrupt location in shared memory. Internal logic within the SCRAMNet interface board prohibits reads and writes from occurring to shared memory simultaneously. Access to shared memory by the local CPU and the network is resolved on a first come first served basis. On a tie condition, the network interface is granted priority. The network control logic consists of several control and status registers. The control registers are used to set the mode of operation for the SCRAMNet board. As mentioned previously certain locations within shared memory can act as interrupt locations (i.e. when they are written to by the network the local CPU is interrupted.) The control logic of the SCRAMNet board allows the programmer to specify which locations, if any, act as interrupt locations. The status registers are monitored during run time to insure that the SCRAMNet is functioning correctly.

Hardware Testbed

In order to evaluate the SCRAMNet real-time shared memory network a three node test bed was setup as shown in Figure 5. Each node consisted of an industry standard Multibus I chassis housing an Intel 80386 20MHz CPU, a Multibus compatible SCRAMNet network interface board, and an Ethernet controller board. Each node ran the Intel iRMXTM II operating system. One of the nodes booted off of its own local hard disk. The other two nodes booted off of a remote hard disk over the Ethernet connection. However, after booting, all three nodes appeared the same to the application software. Each SCRAMNet board contained 128 Kilobytes of shared memory. The nodes were interconnected over a dual channel fiber optic link with 20 meters of spacing between the nodes. Prior

to insertion into the Multibus backplane the shared memory address, the control and status registers address, and the Multibus interrupt were selected via on board jumpers so that no system conflicts would arise. Figure 6 is a simplified depiction of the setup of a node and its attachment to the fiber optic network. Any writes by the local 80386 CPU will pass over the Multibus, through the SCRAMNet host adapter and data filter and then into shared memory. This data is then included in a message packet and transmitted over the network to all of the other nodes. Any incoming network writes are passed through the network interface and written to shared memory. Once in shared memory the data may be read by the local 80386 CPU.

Each SCRAMNet board is equipped with five external triggers which are activated upon different SCRAMNet operations. These operations include the end of a read or write cycle by the host CPU to shared memory, the end of a write to shared memory by the network, the receipt of an interrupt from the network, and the transmission of an interrupt to the network. The triggers were very useful in measuring the transmission delays of data and interrupts transmitted around the network. In the three node testbed one node was chosen to be the sending node. With the aid of an oscilloscope connected to the available triggers it was possible to measure the delays. The timing results are summarized in Table 1 for passing a single 32 bit variable and for passing an interrupt. The timings for the data and interrupt transmission are the same because they are transmitted in the same fashion. The data transmission times only include the time required to transfer data between shared memory on different nodes. It does not include the time required by the local CPU's to perform data writes or reads over the Multibus to shared memory. Also, the interrupt transmission delays listed do not include the time required by the operating system to perform a context switch between tasks. For the iRMX II operating system running on a 20 MHz 80386 CPU an interrupt context switch requires approximately 70 microseconds. The SCRAMNet specifications state that the network can support up to $1.8E+6$ data transfers per second. For the three node network shown in Figure 5, this would be an average of $0.6E+6$ transfers per second from each node or that each node would have to make a write to shared memory every 1.667 microseconds to begin saturating the network. However, the program written to exercise the network's capability could write to shared memory no faster than once every 2.88 microseconds. This is because the CPU must access shared memory over the Multibus and also because the 80386, when operated in virtual protected address mode has no direct way to access a particular physical address location, for example a location in shared memory. Thus, a descriptor (pointer) referring to the beginning of shared memory is created. Creating arrays of

variables based on the created descriptor allows the shared memory to be accessed. Reading and writing arrays indirectly through descriptors requires about 15 times as much CPU time as a direct memory read or write. Thus, even with every node writing data to shared memory as fast as possible, which amounted to a significant amount of network traffic (about $1E+6$ data transfers per second), the network did not become saturated. A typical distributed simulation may not require each node to transfer amounts of data as large as those used in this test. However, it will require that the transmission of smaller sets of data occur very fast and in a deterministic manner. As the number of distributed processors increases or the complexity of the distributed simulation increases the data traffic on the network will also increase. While the SCRAMNet network proved that it could both transmit large amounts of data and do so in a fast and deterministic manner in this particular test, care must be taken in a distributed simulation not to exceed the maximum network bandwidth in order to preserve determinism.

Software Test Application

The NASP Government Baseline Engine program applied the MacCormack numerical method to a Mach 2.5, mixed compression ramjet engine. A schematic of the modelled engine is shown in figure 7. This model had 32 spatial lumps of fixed length and a required a time step of 40 microseconds for real-time operation [2]. The model was run in scaled time on an AD100 computer. When implemented on the AD100 it was determined that this simulation required a frametime of 800 microseconds to perform all the computations which is about 20 times slower than real-time. It was desired to run this code in real-time, but this was not possible on the currently available equipment. To prepare for future programs that would require real-time simulations of this type, we investigated partitioning the simulation and running it on three separate computers as a distributed simulation. The simulation for this model was written in FORTRAN and was first run as a single task on one of the Intel hardware nodes previously discussed as a baseline for comparison. The execution of a single step on one Intel node required 11.4 milliseconds.

The MacCormack method that is used in this simulation is essentially a predictor/corrector integration scheme in both space and time. The model of the system in Figure 7 was manually partitioned into 3 separate tasks along physical lines. We estimated that a reasonably equivalent computational load between the three tasks could be

reached by partitioning the simulation as follows: task 1 is comprised of the inlet and lumps 1-8; task 2 is comprised of lumps 9-20; and task 3 is comprised of the fuel flow and exit nozzle area actuators and lumps 21-32. Since the MacCormack method is a predictor/corrector in space, more data is passed between the lumps than just the information from the adjacent lump. For example, the mass flow calculation at station 10 in task 2 required the mass flow from stations 9 and 8. Going through the code by hand, the necessary variables for the interfaces were extracted. Task 1 and task 3 are each sending 11 and receiving 11 variables upon each update while task 2 is sending 22 and receiving 22 variables on each update. Figure 8 depicts the partitioned simulation and the interfaces between the three nodes. Note that initial conditions had to be supplied for the interface variables in each task in order to start the simulation at an initial trim point. Previously, these variables were calculated in sequence in the unpartitioned simulation. The partitioned code was validated on a VAX minicomputer before porting it to the three nodes of the hardware testbed. The initial conditions for the partitioned model were obtained by running the unpartitioned simulation in single step mode using the VAX Debugger. The interface values were obtained after the first time step was completed. The assumption that was made in partitioning the code is that the one step time delay of the interface variables between the three nodes is not significant.

Partitioned Simulation Results

The partitioned simulation was first run with all three tasks on a single node of the hardware testbed. In other words, the simulation's partitions were executed sequentially on a single CPU. The simulation was coded in FORTRAN and the real-time executive which scheduled its operation was coded in PL/M. An internal timer connected to an interrupt was used to set the update rate for the simulation. The simulation was validated and the time required to execute the simulation was recorded. The results are listed in Table 2. These initial timings provided a benchmark against which to compare the rest of the results. It was noticed while running the partitioned simulation on a single processor that the three tasks did not take the same amount of time to execute. Because of this uneven partitioning, the maximum speedup ratio could never be obtained. It does however point out the importance of partitioning the simulation correctly. Next, the three tasks of the partitioned simulation were implemented on the testbed. One node was interrupted via an internal timer. Upon receiving this interrupt it would immediately generate an interrupt that

was transmitted over the network to the other two nodes. In this way, the update rate of all three nodes was synchronized. After detection of the interrupts, an interrupt handler on each of the three nodes would invoke the local task and also would have detected any update failures had they occurred. All three nodes were set up to operate the following way after being interrupted. First, the necessary inputs were read from shared memory. Next, the partitioned simulation was executed. Finally the outputs were written to shared memory. Processing in this sequence insured that a node would not be attempting to read inputs while the shared memory region was being updated by one of the other nodes. A dynamic comparison of the original and the distributed simulation to a fuel flow step of 5 lbm/sec is shown in Figure 9. The responses for the two cases are identical. The timing for the three tasks are listed in Table 2. It should be pointed out that the execution times listed for the partitioned tasks now include the time required to transmit data through shared memory. Also included in the timings listed for task 2 and task 3 is the time required for the interrupt to propagate around the network and for the context switch to occur. If we take the time required to execute the simulation on one node, 11.40 milliseconds and divide it by the time required for the slowest task to execute in the partitioned three task case, 5.32 milliseconds, we can see that a speedup ratio of 2.14 is obtained. It was estimated that if the simulation had been optimally partitioned, resulting in approximately 4.3 milliseconds of processing time per node, a speedup ratio of 2.65 could have been obtained. A two node partitioned case was also run in which task 1 and task 2 were implemented on one node and task 3 on a second. The timings for this case are also listed in table 2. Since there is less data transferred over the network the total time required for task 1 and task 2 to execute is a little less than the sum of their individual execution times for the three node case. Task 3 remains unchanged so its execution time is the same. The speedup ratio for the two node case is found by dividing the time for the single processor case, 11.40 milliseconds, by the execution time for tasks 1 and 2, 6.84 milliseconds, to arrive at a value of 1.667. It was estimated that optimal partitioning could have improved this factor to a speedup ratio of 1.875. These results will vary depending on the speed of the processor and the time required to access a region of shared memory over the computer bus.

Conclusions

The real-time multi-vendor, multi-computer environment in the Advanced Control Technology Branch at NASA Lewis Research Center requires a real-time communication network. A shared memory network has been proposed as a possible solution to the communication problem, and a candidate shared memory network capable of connecting multiple vendors' computers is described. To test this shared memory network, a model of a ramjet engine was partitioned and simulated on a three node shared memory network. The test results are as follows:

1) the shared memory network is deterministic and the software development for the communication is simple because the hardware handles all the protocols; 2) system models can be partitioned and run on multiple nodes to achieve a speedup in the simulation time, but achieving an optimal speedup requires significant work in terms of partitioning for the target hardware; 3) an investigation of automatic code partitioning software tools is warranted.

References

- 1) Nordwall B.D., "Government-Sponsored Research to Spur Computing and Communications", Aviation Week & Space Technology, June 3, 1991, Computers and Communications Column, p 76.
- 2) Hartley T.T., Melcher K.J., Bruton W.M., "Near Real-Time Large Perturbation Simulation of Internal Fluid Flows", NASP TM-1062, June 1989.
- 3) Bright M.M. and Simon D.L., "The NASA Lewis Integrated Propulsion and Flight Control Simulator", NASA TM 105147, presented at the Flight Simulation Technologies Conference, New Orleans, Louisiana, August 1991.
- 4) Merrill W., Musgrave J.L., and Guo T.H., "Integrated Health Monitoring & Controls for Rocket Engines", SAE-921031, presented at the 1992 SAE Aerospace Atlantic Conference, Dayton, OH, April, 1992.
- 5) Koontz K.W., "Embedded Parallel Architectures in Real-Time Applications", Johns Hopkins APL Technical Digest, Vol. 10, No. 3, 1989, p. 186.
- 6) Rambin R., "Using Ethernet and Fiberoptics (FDDI) Networks in the Real-time Simulation Environment", AIAA 88-53626, presented at the 1988 AIAA Flight Simulation Technologies Conference.
- 7) Shaffer P.L., "A Multiprocessor Implementation of Real-Time Control for a Turbojet Engine", IEEE Control Systems Magazine, Volume 10, Number 4, June 1990, p 38.
- 8) DeLaat J.C. and Merrill W.C., "A Real Time Microcomputer Implementation of Sensor Failure Detection for Turbofan Engines", IEEE Control System Magazine, Volume 10, Number 4, June 1990, p. 29.
- 9) Warden G., "Networking for Real-Time Applications", Systran Corp.
- 10) Jurgensen J.R., "A New Concept in Real-Time Networking", presented at the AIAA Flight Simulation Technology Conference, Sept. 1988.

Table 1 Average Node to Node Transfer Times (Between locations in shared memory)		
	Data Transmission Time (one 32 bit word)	Interrupt Transmission Time
Node 1 Write	0	0
Node 1 Send	0.51 us	0.51 us
Node 2 Receive	1.57 us	1.57 us
Node 3 Receive	1.92 us	1.92 us
Return to Node 1	2.21 us	2.21 us

Table 2 Timing for Distributed Simulation		
Single CPU	Distributed 3 Node Case	Distributed 2 Node Case
Task 1 2.64 ms	Task 1 2.96 ms	Task 1&2 6.84 ms
Task 2 3.88 ms	Task 2 4.62 ms	Task 3 5.32 ms
Task 3 4.88 ms	Task 3 5.32 ms	
<hr/>	<hr/>	<hr/>
Total 11.40 ms	Slowest 5.32 ms	Slowest 6.84 ms
Speedup 1.0 Ratio	Speedup 2.14 Ratio	Speedup 1.67 Ratio

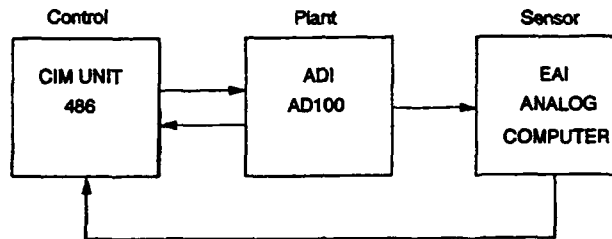


Figure 1.—NASP program hardware configuration.

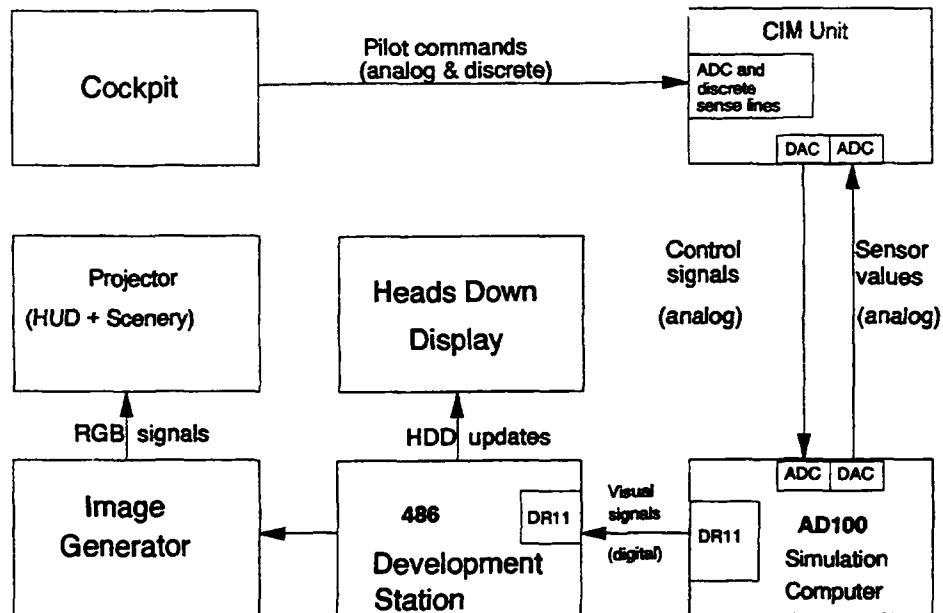


Figure 2.—IFPC program hardware configuration.

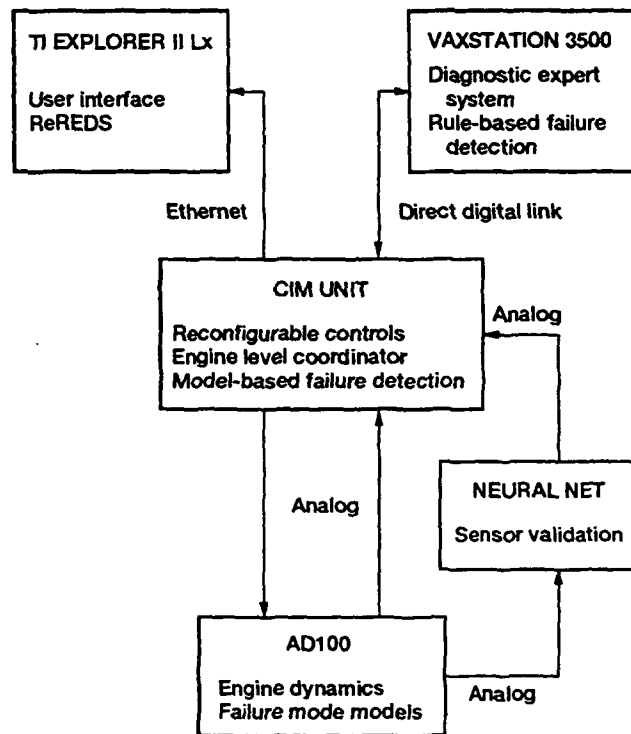


Figure 3.—Intelligent control system simulation test bed.

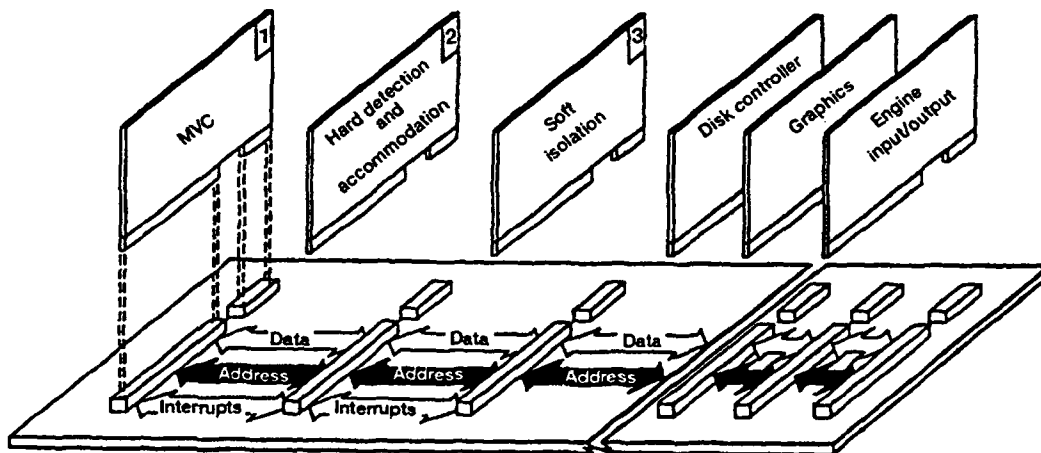


Figure 4.—ADIA parallel processing communication [7].

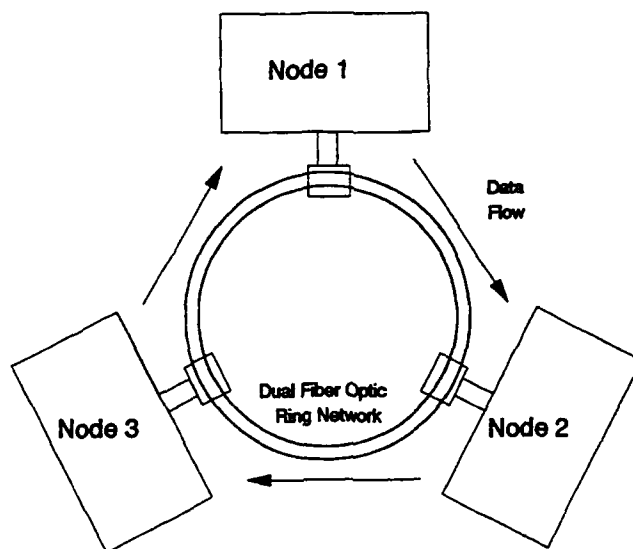


Figure 5.—Three node hardware testbed.

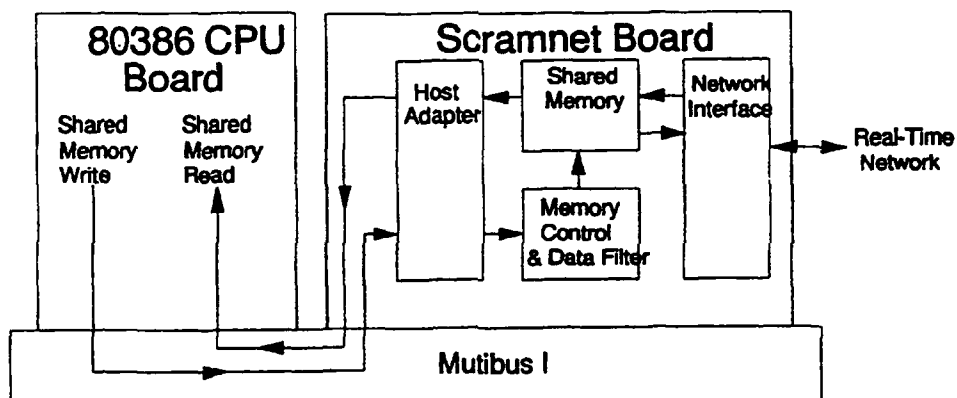


Figure 6.—Multibus scramnet node.

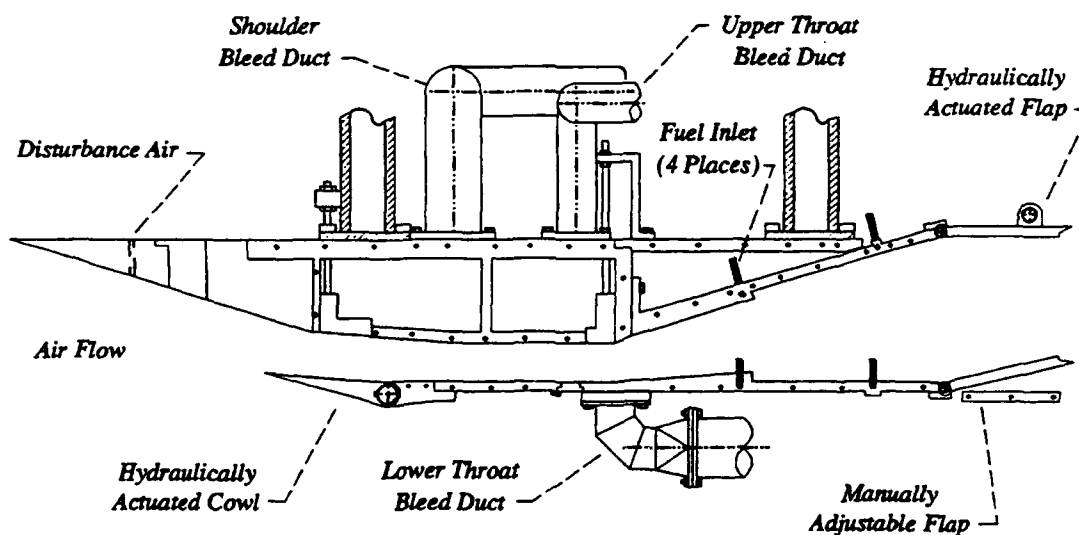


Figure 7.—Ramjet engine schematic.

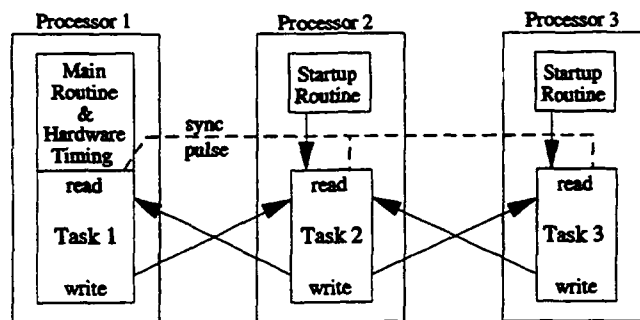


Figure 8.—Partitioned model communication.

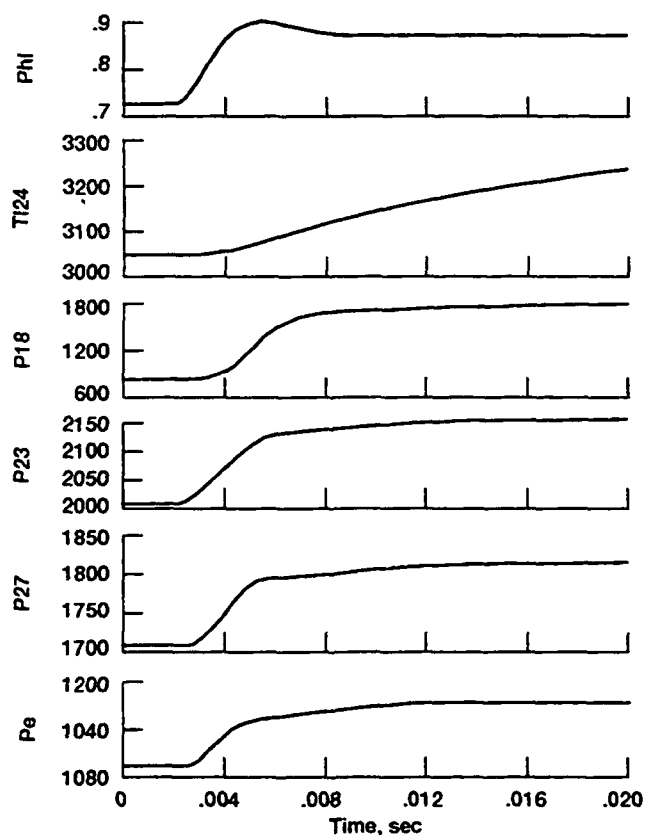


Figure 9.—Comparison of partitioned and nonpartitioned NASP simulations to a 5 lbm fuel flow increase. (Note: Two responses lie on each plot.)

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE July 1993	3. REPORT TYPE AND DATES COVERED Technical Memorandum		
4. TITLE AND SUBTITLE Distributed Simulation Using a Real-Time Shared Memory Network		5. FUNDING NUMBERS WU-505-62-50 1L161102AH45		
6. AUTHOR(S) Donald L. Simon, Duane L. Mattern, Edmond Wong, and Jeffery L. Musgrave				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Lewis Research Center Cleveland, Ohio 44135-3191 and Propulsion Directorate U.S. Army Aviation Research and Technology Activity-AVSCOM Cleveland, Ohio 44135-3127		8. PERFORMING ORGANIZATION REPORT NUMBER E-7969		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546-0001 and U.S. Army Aviation Systems Command St. Louis, Mo. 63120-1798		10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TM-106239 AVSCOM TR-92-C-020		
11. SUPPLEMENTARY NOTES Donald L. Simon, Propulsion Directorate, U.S. Army Aviation Systems Command; Duane L. Mattern, Sverdrup Technology, Inc., Lewis Research Center Group, 2001 Aerospace Parkway, Brook Park, Ohio 44142, and Edmond Wong and Jeffery L. Musgrave, NASA Lewis Research Center. Responsible person, Donald L. Simon, (216) 433-3740.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category 60			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The Advanced Control Technology Branch of the NASA Lewis Research Center performs research in the area of advanced digital controls for aeronautic and space propulsion systems. This work requires the real-time implementation of both control software and complex dynamical models of the propulsion systems. We are implementing these systems in a distributed, multi-vendor computer environment. Therefore, a need exists for real-time communication and synchronization between the distributed multi-vendor computers. A shared memory network is a potential solution which offers several advantages over other real-time communication approaches. In this effort, a candidate shared memory network was tested for basic performance. The shared memory network was then used to implement a distributed simulation of a ramjet engine. The accuracy and execution time of the distributed simulation was measured and compared to the performance of the non-partitioned simulation. The ease of partitioning the simulation, the minimal time required to develop software for communication between the processors and the resulting execution time all indicate that the shared memory network is a real-time communication technique worthy of serious consideration.				
14. SUBJECT TERMS Distributed processing; Real-time operation; Computer networks; Computerized simulation			15. NUMBER OF PAGES 20	
			16. PRICE CODE A03	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT	